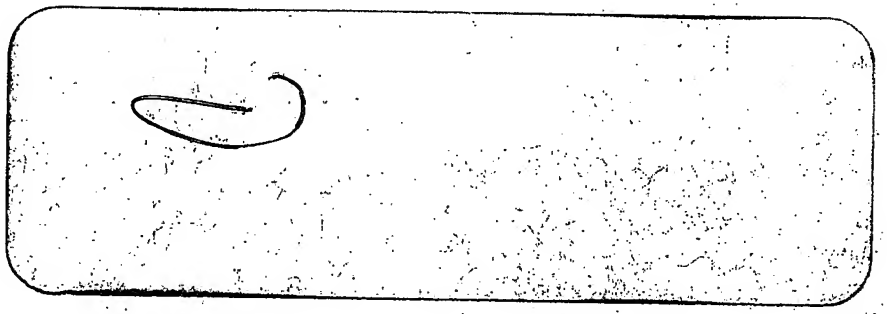
Organization 2100 Bldg/Room KND
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
If Undeliverable Return in Ten Days

OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE, $300

UNITED STATES POSTAGE
PITNEY BOWES
02 1M
0004244975
$ 02.41⁰
FEB 04 2011
MAILED FROM ZIP CODE 22314

AN EQUAL OPPORTUNITY EMPLOYER



RECEIV D
FEB 15 2011
USPTO MAIL CENTER

GRAY155 980042049 1B09 09 02/09/11
FORWARD TIME EXP RTN TO SEND
:GRAYBEAL JACKSON LLP
400 108TH AVE NE STE 700
BELLEVUE WA 98004-8425

RETURN TO SENDER

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

OPAP IAP4
FEB 15 2011
PATENT & TRADEMARK OFFICE

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/684,053	10/09/2003	Chandan Mathur	1934-12-3	3240

7590 02/03/2011

Bryan A. Santarelli
GRAYBEAL JACKSON HALEY LLP
Suite 350
155-108th Avenue NE
Bellevue, WA 98004-5901

EXAMINER	
HUISMAN, DAVID J	

ART UNIT	PAPER NUMBER
2183	

MAIL DATE	DELIVERY MODE
02/03/2011	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/684,053	MATHUR ET AL.	
	Examiner	Art Unit	
	DAVID J. HUISMAN	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *08 October 2010*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-15 and 17-62* is/are pending in the application.

4a) Of the above claim(s) *25-36 and 55-61* is/are withdrawn from consideration.

5)☒ Claim(s) *19-24,51-54 and 62* is/are allowed.

6)☒ Claim(s) *1-15,17,18 and 37-50* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *14 May 2004* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *6/1/2010 & 10/8/2010*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-15 and 17-62 are pending. Claims 25-36 and 55-61 have been withdrawn. Claims 1-15, 17-24, 37-54, and 62 have been examined.

Information Disclosure Statement

2. In the IDS filed on October 8, 2010, NPL document 9 has not been considered (denoted by strike-through) because 37 CFR 1.98(a)(2)(ii) requires that a legible copy be provided. However, the copy provided is very difficult to read and some parts are completely illegible.

Claim Objections

3. Claim 1 is objected to because of the following informalities: In line 4, replace the comma after "to" with a colon. Appropriate correction is required.

4. Claim 42 is objected to because of the following informalities: In line 2, replace "comprise" with --comprises--. Appropriate correction is required.

5. Claim 62 is objected to because of the following informalities: In line 4, it appears that "to", at the end of the line, is followed by a semicolon instead of a colon. Appropriate correction is required. Also, please check other claims for this mistake.

Claim Rejections - 35 USC § 102

6. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –

> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

7. Claims 1-4, 6, 8, 10-13, 15, 18, 37-42, and 45-49 are rejected under 35 U.S.C. 102(e) as being anticipated by Bass et al., U.S. Patent No. 6,405,266 (herein referred to as Bass).

8. Referring to claim 1, Bass has taught a computing machine, comprising:

a) first and second parallel buffers respectively associated with first and second data processing units. See Fig.1 and column 2, lines 18-25. Multiple objects may subscribe to published data. As disclosed, queues/buffers exist between objects for asynchronous communication. For example, in column 6, lines 1-5, object 1 sends data to objects 2 and 4. Hence, a first buffer would exist between objects 1 and 2 and a second buffer would exist between objects 1 and 4.

b) a processor coupled to the buffers (this is deemed inherent, as data is sent from buffers to processor logic for processing) and operable to:

b1) execute an application and first, second, third, and fourth data-transfer objects. All processors inherently execute an application. Also, first through fourth objects are executed as described below.

b2) publish data under the control of the application. An application produces data that is to be sent to a subscribing object by way of a publishing object. For instance, see column 6, lines 1-5. An application publishes a message via a publishing object.

b3) load at least a portion of the published data into the first buffer under the control of the first data-transfer object. See Fig.1 and column 6, lines 1-5. Object 1 (publishing object) will load data into the buffer between it and object 2, for instance, which is a particular subscribing object.

b4) load at least the same portion of the published data into the second buffer under the control of the second data-transfer object. See column 5, lines 52-55. The same data is automatically outputted to the external broker object 103 by message broker object 108 (i.e., the second data-transfer object). Hence, the second data transfer object plays at least some part in loading the same data into the buffer between object 1 and object 4, for instance, which is another particular subscribing object.

b5) retrieve at least the portion of the published data from the first and second buffers under the control of the third and fourth data-transfer object, respectively. See Fig.1, column 3, line 66, to column 4, line 4, and column 6, lines 1-5. The third and fourth objects (i.e., subscribing objects 2 and 4, respectively, retrieve the data asynchronously from the respective buffers). Note also, that one of the third and fourth objects may comprise broker object 113.

9. Referring to claim 2, Bass has taught the computing machine of claim 1 wherein:

a) the first and third data-transfer objects respectively comprise first and second instances of first object code. Note that, in general, the first and third objects comprise message communicating code.

b) the second and fourth data-transfer objects respectively comprise first and second instances of second object code. Similar to above, the second and fourth objects comprise message communicating code.

10. Referring to claim 3, Bass has taught the computing machine of claim 1 wherein the processor comprises:

a) a processing unit operable to execute the application and publish the data under the control of the application. Recall from the rejection of claim 1 that the processor inherently executes an application and publishes data under control of the application. This execution and publishing is performed by a processing unit.

b) a data-transfer handler operable to execute the first, second, third, and fourth data-transfer objects, to load the published data into the first and second buffers under the control of the first and second data-transfer objects, respectively, and to retrieve the published data from the first and second buffers under the control of the third and fourth data-transfer objects, respectively. Again, recall from the rejection of claim 1 that first and second objects loading the first and second buffers, respectively, and third and fourth objects retrieve the data from first and second buffers, respectively. The unit which performs this execution for loading and retrieving data is a data transfer handler.

11. Referring to claim 4, Bass has taught the computing machine of claim 1 wherein the processor is further operable to execute a thread of the application and to publish the data under the control of the thread. See column 3, line 66, to column 4, line 4, and column 5, lines 30-32. Note that the system executes threads.

12. Referring to claim 6, Bass has taught the computing machine of claim 1, further comprising:

a) a bus. This is deemed inherent in a communication system (to carry communications from one location to another).

b) wherein the processor is operable to execute a communication object and to drive the data retrieved from one of the first and second buffers onto the bus under the control of the communication object. See Fig.1, at least component 116, which controls communication.

13. Referring to claim 8, Bass has taught the computing machine of claim 1 wherein the processor is further operable to generate a message that includes a header and data retrieved from one of the first and second buffers under the control of the respective one of the third and fourth data-transfer objects. It should be noted that, as messages are passed between objects 108, 103, and 113, destinations of the messages must be generated (so that it is known which broker to send the message to). Hence, the destination can be considered the header. In addition, even if this were somehow proven untrue (which the examiner does not believe is possible), generating headers for messages is known in the art. Headers provide useful information about the message being transmitted and are used to assist in the communication process. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass such that the processor is further operable to generate a message that includes a header and data retrieved from one of the first and second buffers under the control of the respective one of the third and fourth data-transfer objects. Note, from above, however, that this claim is anticipated.

14. Referring to claim 10, Bass has taught a computing machine, comprising:

a) a first buffer. See column 2, lines 18-25. Multiple objects may subscribe to published data. As disclosed, queues/buffers exist between objects for asynchronous communication. For example, in column 6, lines 1-5, object 1 sends data to objects 2 and 4. Hence, a first buffer would exist between objects 1 and 2. Note also that object 1 may have objects 3 and 4 as

subscribers. Though, this is not explicitly disclosed, one of ordinary skill would realize that any object could subscribe to any particular published item.

b) a processor coupled to the buffer (this is deemed inherent, as data is sent from the buffer to processor logic for processing) and operable to:

b1) execute first and second data-transfer objects and an application.

b2) generate data under control of the application such that the generated data includes no data-destination information. The application inherently generates data. And, as is known with the pub/sub protocol, no data destination information is generated. See column 1, lines 30-34, and column 3, lines 55-58.

b3) retrieve the generated data from the application and load the retrieved data into the buffer under the control of the first data-transfer object. See column 6, lines 1-5. Object 1 gets the data from the application and loads it into a buffer for asynchronous communication.

b4) unload the data from the buffer under the control of the second data-transfer object. See Fig.1, object 2. This second data transfer object, by subscribing to published data of object 1, will read that data from the buffer. Or, the second data transfer object could be object 103, which unloads that data from the first buffer into another buffer for communication with process B (object 4).

b5) process the unloaded data under the control of the application such that the processed data includes only non-data-destination information. Clearly, message data is processed. And, recall that no data destination information is present.

15. Referring to claim 11, Bass has taught the computing machine of claim 10 wherein the first and second data-transfer objects respectively comprise first and second instances of the same object code. Note that, in general, the first and second objects comprise message communicating code. Hence, they are instances of the same object code.

16. Referring to claim 12, Bass has taught the computing machine of claim 10 wherein the processor further comprises:

a) a processing unit operable to execute the application, generate the data, and process the unloaded data under the control of the application. Recall from the rejection of claim 10 that the processor performs each of these claimed functions. They are inherently performed by a processing unit.

b) a data-transfer handler operable to execute the first and second data-transfer objects, to retrieve the data from the application and load the data into the buffer under the control of the first data-transfer object, and to unload the data from the buffer under the control of the second data-transfer object. Again, recall from the rejection of claim 10 that the claimed objects are executed. The unit which performs this execution is a data transfer handler.

17. Referring to claim 13, Bass has taught the computing machine of claim 10 wherein the processor is further operable to execute first and second threads of the application, generate the data under the control of the first thread, and process the unloaded data under the control of the second thread. See column 3, line 66, to column 4, line 4, and column 5, lines 30-32. Note that the system executes threads (e.g., process/thread A and process/thread B in Fig.1). When object 1 communicates with object 4, the first thread generates the data, and the second thread processes the unloaded/sent data.

18. Referring to claim 15, Bass has taught the computing machine of claim 10, further comprising:

a) a second buffer. See column 2, lines 18-25. Multiple objects may subscribe to published data. As disclosed, queues/buffers exist between objects for asynchronous communication. For example, in column 6, lines 1-5, object 1 sends data to objects 2 and 4. Hence, a second buffer would exist between objects 1 and 4.

b) wherein the processor is operable to execute a third data-transfer object, to unload the data from the first buffer into the second buffer under the control of the second data-transfer object, and to provide the data from the second buffer to the application under the control of the third data-transfer object. Again, the second object, under one interpretation (when object 2 is not a subscriber), would be object 103, which would unload data from the first buffer in object 108 and store it in a second buffer, from which object 3 or object 113, for instance, would retrieve it for processing by the application. Object 3 or object 113 may be considered the third data transfer object.

19. Referring to claim 18, Bass has taught the computing machine of claim 10 wherein the processor is further operable to package the generated data into a message that includes a header and the data under the control of the second data-transfer object. It should be noted that, as messages are passed between objects 108, 103, and 113, destinations of the messages must be generated (so that it is known which broker to send the message to). Hence, the destination can be considered the header. In addition, even if this were somehow proven untrue (which the examiner does not believe is possible), generating headers for messages is known in the art. Headers provide useful information about the message being transmitted and are used to assist in

the communication process. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass such that package the generated data into a message that includes a header and the data under the control of the second data-transfer object. Note, from above, however, that this claim is anticipated. Also, note that by simply forwarding the message on (even if it was previously packaged), all of the message data/header is grouped and sent on. So, it is, in a sense, repackaged.

20. Referring to claim 37, Bass has taught a method comprising:

a) publishing data with an application that does not generate an address of a destination of the data. See Fig.1. An application publishes data by way of publishing objects. This is also done without generating a destination address, as is known in the publishing/subscribing protocol. See column 1, lines 30-34, and column 3, lines 55-58.

b) loading the published data into a first buffer with a first data-transfer object, the loaded data absent the address of the destination of the data, each location within the buffer corresponding to the address. See Fig.1 and column 2, lines 18-25. Multiple objects may subscribe to published data. As disclosed, queues/buffers exist between objects for asynchronous communication. For example, in column 6, lines 1-5, object 1 sends data to objects 2 and 4. Hence, a first buffer would exist between objects 1 and 2 or 1 and 4 (or even 1 and 3 if object 3 subscribed to the data published by object 1). Specifically, for published data to get from object 1 to a subscribing object, at least one message broker must be used. Therefore, the buffer is part of the broker. Again, note that an address is not stored in this buffer because the publishing object does not know the destination.

c) retrieving the published data from the buffer with a second data-transfer object, the retrieved data including no information indicating a destination of the data. See Fig.1, column 1, lines 30-34, and column 6, lines 1-5. Note that another object will remove the data from the buffer in order to transmit it to the destination. Again, no address is included in the data.

d) generating a message header that includes a destination of the retrieved data. See Fig.1 and note that the brokers inherently generate addresses since they may send data to multiple locations via publish commands. This address is part of a header.

e) generating a message that includes the retrieved data and the message header. See Fig.1. Broker 103, for instance, sends a message to either broker 108 or 113, or both. Therefore, a destination must be provided with the message.

21. Referring to claim 38, Bass has taught the method of claim 37, wherein publishing the data comprises publishing the data with a thread of the application. See column 3, line 66, to column 4, line 4, and column 5, lines 30-32. Note that the system executes threads.

22. Referring to claim 39, Bass has taught the method of claim 37, further comprising: generating a queue value that corresponds to the presence of the published data in the buffer, notifying the second data-transfer object that the published data occupies the buffer in response to the queue value, wherein retrieving the published data comprises retrieving the published data from the buffer with the second data-transfer object in response to the notification. This is deemed inherent. Clearly, some signal/value must be generated which indicates data is received in a buffer and that it must be retrieved.

23. Referring to claim 40, Bass has taught the method of claim 37, further comprising driving the message onto a bus with a communication object. See Fig.1, at least component 116, which controls communication.

24. Referring to claim 41, Bass has taught the method of claim 37, further comprising loading the retrieved data into a second buffer with the second data-transfer object. Each broker has at least one buffer to store message sent to it. Therefore, second data-transfer object 108 would load data from the first buffer into a buffer in broker 103. Or, second data transfer object 103 would load data from the first buffer into a buffer in broker 113.

25. Referring to claim 42, Bass has taught the method of claim 37 wherein generating the message header and the message comprise generating the message header and the message with the second data transfer object. See Fig.1 and note the messages with headers are generated when they are passed between brokers. The initial sources and ultimate destinations know nothing about the addresses, as is known in the pub/sub protocol.

26. Referring to claim 45, Bass has taught a method comprising:

a) receiving a message that includes data and that includes a message header that indicates a destination address of the data, the destination address corresponding to a software application. See Fig.1. Message are passed between processes/applications on the same or different processors (column 6, lines 40-49). As discussed throughout Bass, and in column 1, lines 21-33, process objects communicate by way of a pub/sub protocol through a series of brokers. As is known, the initial object that publishes the data has no knowledge of the ultimate destination address. Similarly, the ultimate destination has no idea of the sender's address. However, it is inherent, as messages are passed between brokers, that addresses in headers accompany them.

For instance, broker 103 can send to one or more processes/applications, and therefore, must know which address to send a message to. Hence, broker 103 will retrieve a message from one process and attach an address of another process broker 108/113.

b) loading into a first buffer with a first data-transfer object, the received data without the message header, the first buffer corresponding to the destination. See Fig.1 and column 2, lines 18-25. Multiple objects may subscribe to published data. As disclosed, queues/buffers exist between objects for asynchronous communication. For example, in column 6, lines 1-5, object 1 sends data to objects 2 and 4. Hence, a first buffer would exist to hold the message sent from broker 103 to object 4. Note that the data and the header are separate so, the data is stored without the header. Also, it is known that a destination is stripped when it arrives to the destination. Storing it is generally unnecessary and would require additional memory to do so.

c) unloading the data from the buffer with a second data-transfer object and processing the unloaded data with an application corresponding to the destination.. See Fig.1 and column 6, lines 1-5. A second object (either broker 113 or destination object 4) will unload the data from the buffer for processing by the application process.

27. Referring to claim 46, Bass has taught the method of claim 45 wherein processing the unloaded data comprises processing the unloaded data with a thread of the application corresponding to the destination. See column 3, line 66, to column 4, line 4, and column 5, lines 30-32. Note that the system executes threads.

28. Referring to claim 47, Bass has taught the method of claim 45, further comprising: generating a queue value that corresponds to the presence of the data in the buffer, notifying the second data-transfer object that the published data occupies the buffer in response to the queue

value, wherein unloading the data comprises unloading the data from the buffer with the first data-transfer object in response to the notification. This is deemed inherent. Clearly, some signal/value must be generated which indicates data is received in a buffer and that it must be retrieved/unloaded. Broker 113 would then send the data to broker 113.

29. Referring to claim 48, Bass has taught the method of claim 45, further comprising wherein receiving the message comprises receiving the message with the first data-transfer object. See Fig.1. Broker 103 receives the data.

30. Referring to claim 49, Bass has taught the method of claim 45, further comprising: receiving the message comprises retrieving the message from a bus with a communication object and transferring the data from the communication object to the first data transfer object. See Fig.1. Communication object 108 or 116 retrieves the data from the initial source and sends it on to broker 103.

Claim Rejections - 35 USC § 103

31. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

32. Claim 7 is rejected under 35 U.S.C. 103(a) as being unpatentable over Bass.

33. Referring to claim 7, Bass has taught the computing machine of claim 1. Bass has not taught a third buffer. However, one of ordinary skill in the art would have recognized that there may be more than just two subscribing objects. Pub/sub allows for scalability and for more

subscribers to exist if desired. Hence, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass to include another object (either in process B or in another process (e.g., process C) which subscribes to publishings by object 1 via a third buffer. Then, Bass, as modified, has taught that the processor is operable to provide the data retrieved from one of the first and second buffers to the third buffer under the control of the respective one of the third and fourth data-transfer objects. See Fig.1. If the new object is part of process B then third/fourth object 113 would transfer the data from the first buffer managed by process A into the third buffer managed by process B.

34. Claims 5, 9, 14, 17, 43-44, and 50 are rejected under 35 U.S.C. 103(a) as being unpatentable over Bass in view of the examiner's taking of Official Notice.

35. Referring to claim 5, Bass has taught the computing machine of claim 1. Bass has further taught that the processor is further operable to:

a) store a queue value, the queue value reflecting the loading of the published data into the first buffer, read the queue value, notify the third data-transfer object that the published data occupies the first buffer in response to the queue value, and retrieve the published data from the first buffer under the control of the third data-transfer object and in response to the notification. This is deemed inherent. Clearly when data is received, some value must be generated and read to indicate and notify that data is available in the queue for transmission/retrieval.

b) Bass has not taught executing a queue object and a reader object, the queue value is stored under the control of the queue object, the queue value is read under the control of the reader object, and the third data-transfer object is notified that the published data occupies the first

buffer under the control of the reader object. However, since Bass's environment is an object-oriented environment, one of ordinary skill in the art would have recognized that components may be programmed as software objects, including the components which generate and read the queue values and/or perform notification. Since software may be quickly modified to change functionality, design of these components is more flexible compared to dedicated hardware components for performing the claimed functions. If a problems exists in the hardware, it must be rebuilt instead of reprogrammed, which is more difficult and time-consuming. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass to execute a queue object and a reader object, to store the queue value under the control of the queue object, to read the queue value under the control of the reader object, and to notify the third data-transfer object that the published data occupies the first buffer under the control of the reader object.

36. Referring to claim 9, Bass has taught the computing machine of claim 1 wherein:

a) the first and third data-transfer objects respectively comprise first and second instances of first object code. Note that, in general, the first and third objects comprise message communicating code.

b) the second and fourth data-transfer objects respectively comprise first and second instances of second object code. Similar to above, the second and fourth objects comprise message communicating code.

c) Bass has not taught that the processor is operable to execute an object factory and to generate the first object code and the second object code under the control of the object factory. However, object factories and their advantages are known in the art of object oriented

programming. Specifically, an object factory is an object used to create other objects, and has proven advantages. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass such that the processor is operable to execute an object factory and to generate the first object code and the second object code under the control of the object factory.

37. Referring to claim 14, Bass has taught the computing machine of claim 10. Bass has further taught that the processor is further operable to:

a) store a queue value, the queue value reflecting the loading of the retrieved data into the first buffer, read the queue value, notify the second data-transfer object that the retrieved data occupies the buffer in response to the queue value, and unload the retrieved data from the buffer under the control of the second data-transfer object and in response to the notification. This is deemed inherent. Clearly when data is received, some value must be generated and read to indicate and notify that data is available in the queue for transmission/retrieval.

b) Bass has not taught executing a queue object and a reader object, the queue value is stored under the control of the queue object, the queue value is read under the control of the reader object, and the second data-transfer object is notified that the retrieved data occupies the buffer under the control of the reader object. However, since Bass's environment is an object-oriented environment, one of ordinary skill in the art would have recognized that components may be programmed as software objects, including the components which generate and read the queue values and/or perform notification. Since software may be quickly modified to change functionality, design of these components is more flexible compared to dedicated hardware components for performing the claimed functions. If a problems exists in the hardware, it must

be rebuilt instead of reprogrammed, which is more difficult and time-consuming. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass to execute a queue object and a reader object, to store the queue value under the control of the queue object, to read the queue value under the control of the reader object, and to notify the second data-transfer object that the retrieved data occupies the buffer under the control of the reader object.

38. Referring to claim 17, Bass has taught the computing machine of claim 10 wherein:

a) the first and second data-transfer objects respectively comprise first and second instances of the same object code. Note that, in general, the first and second objects comprise message communicating code. Hence, they are instances of the same object code.

b) Bass has not taught that the processor is operable to execute an object factory and to generate the object code under the control of the object factory. However, object factories and their advantages are known in the art of object oriented programming. Specifically, an object factory is an object used to create other objects, and has proven advantages. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass such that the processor is operable to execute an object factory and to generate the object code under the control of the object factory.

39. Referring to claim 43, Bass has taught the method of claim 37, further comprising:

a) generating the first data-transfer object as a first instance of data-transfer object code and generating the second data-transfer object as a second instance of the object code. Note that, in general, the first and second objects comprise message communicating code. Therefore, they are instances of communication object code.

b) Bass has not taught generating the object code with an object factory. However, object factories and their advantages are known in the art of object oriented programming. Specifically, an object factory is an object used to create other objects, and has proven advantages. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass such that the processor is operable to execute an object factory and to generate the object code under the control of the object factory.

40. Referring to claim 44, Bass has taught the method of claim 37. Bass has not taught receiving the message and processing the data in the message with a hardwired pipeline accelerator. However, Official Notice is taken that hardwired pipelined processors and their advantages are well known and accepted in the art. A pipeline allows for the overlapping of execution, instead of serial execution, thereby speeding up the processor and increasing throughput. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass's receiving processor such that it includes a hardwired pipeline for accelerating execution.

41. Referring to claim 50, Bass has taught the method of claim 45. Bass has not explicitly taught generating the message header and the message with a hardwired pipeline accelerator. However, Official Notice is taken that hardwired pipelined processors and their advantages are well known and accepted in the art. A pipeline allows for the overlapping of execution, instead of serial execution, thereby speeding up the processor and increasing throughput. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass's processor executing broker code 103 such that it includes a hardwired pipeline for accelerating execution.

42. Claims 10-15, 17-18, and 45-50 are rejected under 35 U.S.C. 103(a) as being unpatentable over Dretzka et al., U.S. Patent No. 4,703,475 (herein referred to as Dretzka), in view of the examiner's taking of Official Notice. Note that some of the claims are rejected multiple times under different interpretations of Dretzka.

43. Referring to claim 10, Dretzka has taught a computing machine, comprising:

a) a first buffer. See Fig.6, buffer 220-4, for instance.

b) a processor (Fig.1, component 21) coupled to the buffer and operable to:

b1) execute first and second data-transfers and an application. Fig.4 sets forth at least some of the data-transfers executed by processor 21. Looking at Fig.4 and Fig.6, a first transfer would be executed to load data into buffer 220-4 and a second transfer would be executed to unload data from buffer 220-4. At least some of the other functions performed by the processor (for instance, doing normal ALU operations) would be part of the application inherently executed by the processor.

b2) generate data under control of the application without generating an address of a destination of the data. See Fig.6 and Figs.8-15, and note that the level 2.5 application generates data using an input list by searching for consecutive sequence packets. Also, note from Fig.4, at level 2.5, the 1-byte header (data-destination information) is deleted, and therefore, the data includes no data-destination information. In an alternate interpretation, if an address is inherently generated in Dretzka, as applicant argues, the address may still be considered as being separate from the generated data (i.e., the true

content of the message). Therefore, the generated data itself does not include any data-destination information and, further, the data is generated without generating the address.

b3) retrieve the generated data from the application and load the retrieved data into the buffer under the control of the first data-transfer. See Fig.6. After data is generated by the input list, the first transfer transfers it to the buffer (i.e., 220-4).

b3) unload the data from the buffer under the control of the second data-transfer. See Fig.6. Data is ultimately moved/unloaded from the buffer 220-4 and into buffer 210 under control of the second transfer.

b4) process the unloaded data under the control of the application without receiving the address of the destination of the data. See Fig.4 and Fig.6. From the buffer 210, the processor will process the data using an application. Also, note from Fig.4, at level 2.5 (which occurs well before the unloading), the 1-byte header (data-destination information) is deleted, and therefore, the processed data includes no data-destination information. In the alternate interpretation, if an address inherently exists, as argued by applicant, then this address is still separate from the generated data. Hence, data is processed separately from the address. That is the address does not need to be received to process the data.

b5) Dretzka has not taught executing first and second data-transfer objects. However, object-oriented programming and objects are known and have known advantages in the art. Therefore, particular features of Dretzka may be implemented as objects. For instance, a loader for loading data into a buffer may be implemented as an object. Likewise, an unloader for unloading data from a buffer may be implemented as an object.

Since object-oriented programming has known benefits to at least some programmers, such as making programs easier to manage and keep track of, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor executes first and second data transfer objects to carry out the respective transfers.

44. Referring to claim 11, Dretzka, as modified, has taught the computing machine of claim 10 wherein the first and second data-transfer objects respectively comprise first and second instances of the same object code. See Fig.6 and note that, in general, the first object retrieves data from a previous stage (level 2.5 stage) and stores it in a next buffer 220-4. Likewise, the second object retrieves data from a previous stage (level 3) and stores it in a next buffer 210. Hence, both of these objects comprise instances of general "retrieve-and-store" object code.

45. Referring to claim 12, Dretzka, as modified, has taught the computing machine of claim 10 wherein the processor further comprises:

a) a processing unit operable to execute the application, generate the data, and process the unloaded data under the control of the application. Recall from the rejection of claim 10 (and from Fig.4) that the processor performs each of these claimed functions. They are inherently performed by a processing unit.

b) a data-transfer handler operable to execute the first and second data-transfer objects, to retrieve the data from the application and load the data into the buffer under the control of the first data-transfer object, and to unload the data from the buffer under the control of the second data-transfer object. Again, recall from the rejection of claim 10 that the claimed objects are executed. The unit which performs this execution is a data transfer handler.

46. Referring to claim 13, Dretzka, as modified, has taught the computing machine of claim 10. Dretzka has not taught that the processor is further operable to execute first and second threads of the application, to generate the data under the control of the first thread, and to process the unloaded data under the control of the second thread. However, Official Notice is taken that multithreaded processors and their advantages are well known and accepted in the art. Specifically, it is known to divide up a program into threads in order to increase efficiency by reducing stall time. With multiple threads, since threads are independent sequences of instructions, the system may switch to a next thread when a first thread stalls, thereby hiding the stall time require by the first thread. Essentially, the processor is kept busy as often as possible with multithreading. As a result, in order to increase efficiency, and because generating data and processing unloaded data are independent tasks, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor is operable to execute first and second threads of the application, to generate the data under the control of the first thread, and to process the unloaded data under the control of the second thread.

47. Referring to claim 14, Dretzka, as modified, has taught the computing machine of claim 10 wherein the processor is further operable to:

a) execute a queue object and a reader object. See Fig.4 and Fig.6. The queue object is the object which stores the more bit, which ultimately leads to the informing of level 4 that a complete message has been received. The reader object is the object which detects the more bit so that further action can occur.

b) store a queue value under the control of the queue object, the queue value reflecting the loading of the retrieved data into the first buffer. See Fig.4 and Fig.6. The queue object will

store the "more" bit, which reflects the loading of the retrieved data into the buffer. This bit, when set in a certain manner, will allow for the informing of level 4 of a complete message.

c) read the queue value under the control of the reader object. See Fig.4 and note that the more bit, when set to a certain level, will indicate the end of the message and that the message may be further transmitted and processed. The reader object will have to read this more bit.

d) notify the second data-transfer object that the retrieved data occupies the buffer under the control of the reader object and in response to the queue value. When the "more" bit is set in the appropriate manner and noted by the reader object, the data may be transferred to the next-level buffer. See Fig.4 and Fig.6.

e) unload the retrieved data from the buffer under the control of the second data-transfer object and in response to the notification. Again, in response to the notification, the data will be moved from level 3 buffer to level 4 buffer. See Fig.4 and Fig.6.

f) Note that while queue and reader objects are not explicitly recited in Dretzka, implementing the queue/reader functionality as objects for execution is obvious for the reasons set forth in claim 10.

48. Referring to claim 15, Dretzka, as modified, has taught the computing machine of claim 10, further comprising:

a) a second buffer. See Fig.6, component 210.

b) wherein the processor is operable to execute a third data-transfer object, to unload the data from the first buffer into the second buffer under the control of the second data-transfer object, and to provide the data from the second buffer to the application under the control of the third data-transfer object. See Fig.4 and Fig.6. Data is unloaded from buffer 220-4 to buffer 210

under control of the second transfer object, and then moved from buffer 210 to the application in the processor under control of the third object.

c) Note that while the third object is not explicitly recited in Dretzka, implementing the third data transfer via object execution is obvious for the reasons set forth in claim 10.

49. Referring to claim 17, Dretzka, as modified, has taught the computing machine of claim 10 wherein:

a) the first and second data-transfer objects respectively comprise first and second instances of the same object code. See Fig.6 and note that, in general, the first object retrieves data from a previous stage (level 2.5 stage) and stores it in a next buffer 220-4. Likewise, the second object retrieves data from a previous stage (level 3) and stores it in a next buffer 210. Hence, both of these objects comprise instances of general "retrieve-and-store" object code.

b) Dretzka has not taught that the processor is operable to execute an object factory and to generate the object code under the control of the object factory. However, object factories and their advantages are known in the art of object oriented programming. Specifically, an object factory is an object used to create other objects, and has proven advantages. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor is operable to execute an object factory and to generate the object code under the control of the object factory.

50. Referring to claim 10, Dretzka has taught a computing machine (under a second interpretation), comprising:

a) a first buffer. See Fig.5, buffer 110, for instance.

b) a processor (Fig.1, component 11) coupled to the buffer and operable to:

b1) execute first and second data-transfers and an application. Fig.3 sets forth at least some of the data-transfers executed by processor 11. Looking at Fig.3 and Fig.5, a first transfer would be executed to load data into buffer 110 and a second transfer would be executed to unload data from buffer 110 and into buffer 120-4, for instance. At least some of the other functions performed by the processor (for instance, doing normal ALU operations) would be part of the application inherently executed by the processor.

b2) generate data under control of the application without generating an address of a destination of the data. See Fig.5 and note that a message comprising data must inherently be generated before being stored in buffer 110. This data, as shown at the top of Fig.3, is generated as a result of application execution. Note that at this time of generation, no headers and/or data-destination information are attached to the generated data. In an alternate interpretation, if an address is inherently generated in Dretzka, as applicant argues, the address is separate from generated data. Therefore, the generated data itself does not include any data-destination information and, further, the data is generated without generating the address.

b3) retrieve the generated data from the application and load the retrieved data into the buffer under the control of the first data-transfer. See Fig.5. After data is generated, it is ultimately stored in buffer 110.

b3) unload the data from the buffer under the control of the second data-transfer. See Fig.5. Data is ultimately moved/unloaded from the buffer 110 and into buffer 120-4 under control of the second transfer.

b4) process the unloaded data under the control of the application without receiving the address of the destination of the data. See Fig.3 and note that the data, after being moved into buffer 120-4, is further processed by breaking the message up. It isn't until the data is in the next buffer that a 1-byte channel header, which may or may not be considered data-destination information is added to it. Hence, at the time of the claimed processing, since the 1-byte header has not been added, the data does not include data-destination information. In the alternate interpretation, if an address inherently exists, as argued by applicant, then this address is still separate from the generated data. Hence, data is processed separately from the address. That is the address does not need to be received to process the data.

b5) Dretzka has not taught executing first and second data-transfer objects. However, object-oriented programming and objects are known and have known advantages in the art. Therefore, particular features of Dretzka may be implemented as objects. For instance, a loader for loading data into a buffer may be implemented as an object. Likewise, an unloader for unloading data from a buffer may be implemented as an object. Since object-oriented programming has known benefits to at least some programmers, such as making programs easier to manage and keep track of, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor executes first and second data transfer objects to carry out the respective transfers.

51. Referring to claim 11, Dretzka, as modified, has taught the computing machine of claim 10 (under a second interpretation) wherein the first and second data-transfer objects respectively

comprise first and second instances of the same object code. See Fig.5 and note that, in general, the first object retrieves data from a source and stores it in a next buffer 110. Likewise, the second object retrieves data from a source (buffer 110) and stores it in a next buffer 120-4. Hence, both of these objects comprise instances of general "retrieve-and-store" object code.

52. Referring to claim 12, Dretzka, as modified, has taught the computing machine of claim 10 (under a second interpretation) wherein the processor further comprises:

a) a processing unit operable to execute the application, generate the data, and process the unloaded data under the control of the application. Recall from the rejection of claim 10 (and from Fig.3) that the processor performs each of these claimed functions. They are inherently performed by a processing unit.

b) a data-transfer handler operable to execute the first and second data-transfer objects, to retrieve the data from the application and load the data into the buffer under the control of the first data-transfer object, and to unload the data from the buffer under the control of the second data-transfer object. Again, recall from the rejection of claim 10 that the claimed objects are executed. The unit which performs this execution is a data transfer handler.

53. Referring to claim 13, Dretzka, as modified, has taught the computing machine of claim 10. Dretzka has not taught that the processor is further operable to execute first and second threads of the application, to generate the data under the control of the first thread, and to process the unloaded data under the control of the second thread. However, Official Notice is taken that multithreaded processors and their advantages are well known and accepted in the art. Specifically, it is known to divide up a program into threads in order to increase efficiency by reducing stall time. With multiple threads, since threads are independent sequences of

instructions, the system may switch to a next thread when a first thread stalls, thereby hiding the stall time require by the first thread. Essentially, the processor is kept busy as often as possible with multithreading. As a result, in order to increase efficiency, and because generating data and processing unloaded data are independent tasks, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor is operable to execute first and second threads of the application, to generate the data under the control of the first thread, and to process the unloaded data under the control of the second thread.

54. Referring to claim 14, Dretzka, as modified, has taught the computing machine of claim 10 (under a second interpretation).

a) Dretzka has not taught that the processor is further operable to execute a queue object and a reader object. However, recall from Fig.5 that data is initially stored in a queue. The examiner asserts that it is well known and advantageous to have an empty bit indicate the status of the queue because it prevents the queue from being read if it has no useful data in it, thereby saving time. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka to execute a queue object to set/clear an empty bit based on whether the queue contains valid data to be read, and also a reader object to read the empty bit such that the system knows when to read the valid data from the queue.

b) Dretzka, as modified, has further taught storing a queue value under the control of the queue object, the queue value reflecting the loading of the retrieved data into the first buffer. This is deemed inherent as an empty bit would be stored.

c) Dretzka, as modified, has further taught reading the queue value under the control of the reader object. Again, this is deemed inherent because if an empty bit were implemented, it would be meant for reading so that the system knows when the queue is empty.

d) Dretzka, as modified, has further taught notifying the second data-transfer object that the retrieved data occupies the buffer under the control of the reader object and in response to the queue value. When the empty bit is clear and noted by the reader object, the data exists in the queue and should be handled.

e) Dretzka, as modified, has further taught unloading the retrieved data from the buffer under the control of the second data-transfer object and in response to the notification. Again, in response to the notification, the data will be moved from buffer 110 to 120-4.

f) Finally, note that while Dretzka hasn't recited objects, as claimed, implementing the above functions via object execution is obvious for the reasons set forth in claim 10.

55. Referring to claim 15, Dretzka, as modified, has taught the computing machine of claim 10 (under a second interpretation), further comprising:

a) a second buffer. See Fig.5, component 120-4.

b) wherein the processor is operable to execute a third data-transfer object, to unload the data from the first buffer into the second buffer under the control of the second data-transfer object, and to provide the data from the second buffer to the application under the control of the third data-transfer object. See Fig.3 and Fig.5. Data is unloaded from buffer 110 to buffer 120-4 (second buffer) under control of the second transfer object, and then moved from buffer 120-4 to an additional buffer and interface under control of the third object.

c) Note that while the third object is not explicitly recited in Dretzka, implementing the third data transfer via object execution is obvious for the reasons set forth in claim 10.

56. Referring to claim 17, Dretzka, as modified, has taught the computing machine of claim 10 (under a second interpretation) wherein:

a) the first and second data-transfer objects respectively comprise first and second instances of the same object code. See Fig.5 and note that, in general, the first object retrieves data from a source and stores it in a next buffer 110. Likewise, the second object retrieves data from a source (buffer 110) and stores it in a next buffer 120-4. Hence, both of these objects comprise instances of general "retrieve-and-store" object code.

b) the processor is operable to execute an object factory and to generate the object code under the control of the object factory. All processors execute programs. The program (object factory) will dictate when data needs to be transmitted and received. That is, when the program calls for data to be transmitted, the first and second object codes will be generated and invoked so that data may be transmitted.

57. Referring to claim 18, Dretzka, as modified, has taught the computing machine of claim 10 (under a second interpretation) wherein the processor is further operable to package the generated data into a message that includes a header and the data under the control of the second data-transfer object. See Fig.3 (at least the level 3 object code), and note that the second object begins packaging the data into a message.

58. Referring to claim 45, Dretzka has taught a method comprising:

a) receiving a message that includes data and that includes a message header that indicates a destination address of the data, the destination address corresponding to a software application. See Fig.4 and Fig.6. Note that a message comes in with a 1-byte header (note level 2.5) that will indicate, upon review by the receiving end, a destination of either input list 230-x or message buffer 220-x (column 7, line 25, to column 8, line 31). Clearly, the message must be received by some corresponding application.

b) loading into a first buffer with via first data-transfer, the received data without the message header, the first buffer corresponding to the destination. See Fig.6 and note that the data is loaded into buffer 220-4 based on the 1-byte header. Note that the 1-byte message header is removed prior to storage in 220-4 according to Fig.4.

c) unloading the data from the buffer with via second data-transfer. See Fig.4 and Fig.6 and note that data is ultimately unloaded from buffer 220-4.

d) processing the unloaded data with an application corresponding to the destination. See Fig.4 and note that after the data is unloaded, it will be sent to the processor where inherent processing on that data will commence.

e) Dretzka has not taught executing first and second data-transfer objects. However, object-oriented programming and objects are known and have known advantages in the art. Therefore, particular features of Dretzka may be implemented as objects. For instance, a loader for loading data into a buffer may be implemented as an object. Likewise, an unloader for unloading data from a buffer may be implemented as an object. Since object-oriented programming has known benefits to at least some programmers, such as making programs easier to manage and keep track of, it would have been obvious to one of ordinary skill in the art at the time of the invention to

modify Dretzka such that the processor executes first and second data transfer objects to carry out the respective transfers.

59. Referring to claim 46, Dretzka, as modified, has taught the method of claim 45. Dretzka has not taught that processing the unloaded data comprises processing the unloaded data with a thread of the application corresponding to the destination. However, Official Notice is taken that multithreaded processors and their advantages are well known and accepted in the art. Specifically, it is known to divide up a program into threads in order to increase efficiency by reducing stall time. With multiple threads, the system may switch to a second thread when a first thread stalls, thereby hiding the stall time require by the first thread. Essentially, the processor is kept busy as often as possible with multithreading. As a result, in order to increase efficiency, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor processes the unloaded data with a thread.

60. Referring to claim 47, Dretzka, as modified, has taught the method of claim 45, further comprising:

a) generating a queue value that corresponds to the presence of the data in the buffer. See Fig.4 and Fig.6. The queue object will generates the "more" bit, which corresponds to the presence of data in the buffer. This bit, when set in a certain manner, will allow for the informing of level 4 of a complete message.

b) notifying the second data-transfer object that the data occupies the buffer in response to the queue value. When the "more" bit is set in the appropriate manner and noted by the reader object, the data may be transferred to the next-level buffer by informing (notifying) the next level that data is ready to be transferred. See Fig.4 and Fig.6.

c) wherein unloading the data comprises unloading the data from the buffer with the first data-transfer object in response to the notification. Again, in response to the notification, the data will be moved from level 3 buffer to level 4 buffer. See Fig.4 and Fig.6.

61. Referring to claim 48, Dretzka, as modified, has taught the method of claim 45, further comprising wherein receiving the message comprises receiving the message with the first data-transfer object. See Fig.4 and Fig.6. Some code/object will cause the receiving of the data. That object is the first data transfer object. Note that while the communication object is not explicitly recited in Dretzka, implementing the receiving via object execution is obvious for the reasons set forth in claim 45.

62. Referring to claim 49, Dretzka, as modified, has taught the method of claim 45, further comprising:

a) receiving the message comprises retrieving the message from a bus with a communication object. See Fig.4 and Fig.6. Some code/object will cause the receiving of the data from a bus. That object is the communication object. Note that while the communication object is not explicitly recited in Dretzka, implementing the receiving via object execution is obvious for the reasons set forth in claim 45.

b) transferring the data from the communication object to the first data transfer object. See Fig.4 and Fig.6. Note that after the data is received, it is passed to the first data transfer object which at least stores it in buffer 220-4.

63. Referring to claim 50, Dretzka, as modified, has taught the method of claim 45. Dretzka has not explicitly taught generating the message header and the message with a hardwired pipeline accelerator. However, Official Notice is taken that hardwired pipelined processors and

their advantages are well known and accepted in the art. A pipeline allows for the overlapping of execution, instead of serial execution, thereby speeding up the processor and increasing throughput. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka's processor (Fig.1, component 11) such that it includes a hardwired pipeline for accelerating execution. Note that in this series of rejections, processor 11 is the unit which generates the messages according to Fig.3.

Allowable Subject Matter

64. Claims 19-24, 51-54, and 62 are allowed. Please review all claims for informalities.

65. The examiner asserts that the term "object" in the allowable (and non-allowable) claims is hereby limited to an object as is known in the art of object-oriented programming. "Object" cannot be interpreted as anything but an object-oriented object, which binds data with methods that operate on that data. See page 27 of applicant's arguments. Variations of "object", e.g., "objects" are limited in the same fashion.

66. The examiner asserts that the term "publish" in the allowable (and non-allowable) claims is hereby limited to the publish action as is known in the art of publish/subscribe messaging protocol(s). "Publish" cannot be interpreted as anything but what is consistent in publish/subscribe communication. Specifically, to publish is to transmit data to one or more destinations from a source where the source does not know the destination or generate the address of the destination. See page 28 of applicant's arguments. Variations of "publish", e.g., "publishing", "published", etc., are limited in the same fashion.

Response to Arguments

67. Arguments pertaining to claims no longer rejected under Dretzka are hereby moot and will not be addressed by the examiner.

68. With respect to the argument of the rejection of claims 10 and 37:

a) the examiner asserts that even if a destination address must be generated (to specify one of modules 10, 20, and 30 for communication in Fig.1), this address may still be considered as being separate from the generated data (e.g., the payload) and, therefore, the generated data does not include data destination information. That is, the message data and the address of the component to which the message data will be sent are distinct. Furthermore, please note that the examiner still maintains his previous interpretation as well (regarding the 1 and 3-byte headers).

b) the examiner admits that Dretzka has not taught "objects" as applicant has argued. However, the examiner asserts that it would have been obvious for Dretzka's transfers to occur via object execution due to the well-known advantages of object-oriented programming. See the new ground of rejection above.

c) the examiner admits that Dretzka has not taught "publishing" as applicant has argued. Therefore, the rejection of claim 37 under Dretzka has been withdrawn. However, note the new grounds of rejection under Bass for claim 37.

69. With respect to the argument of the rejection of claim 45:

a) the examiner admits that Dretzka has not taught "objects" as applicant has argued. However, the examiner asserts that it would have been obvious for Dretzka's transfers to

occur via object execution due to the well-known advantages of object-oriented programming. See the new ground of rejection above.

b) the examiner asserts that since it is obvious to modify Dretzka to be in an object-oriented environment, the it follows that the messages in the system must be compatible with an object interface.

Conclusion

70. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL.** See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

The prior art made of record and not relied upon is considered pertinent to applicant's disclosure. Applicant is reminded that in amending in response to a rejection of claims, the patentable novelty must be clearly shown in view of the state of the art disclosed by the

references cited and the objections made. Applicant must also show how the amendments avoid such references and objections. See 37 CFR § 1.111(c).

Oki et al., "The Information Bus - An Architecture for Extensible Distributed Systems", 1993, pp.58-68, discusses the publish/subscribe protocol.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to DAVID J. HUISMAN whose telephone number is (571)272-4168. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/David J. Huisman/
Primary Examiner, Art Unit 2183

Receipt date: 06/01/2010

Doc code: IDS

Doc description: Information Disclosure Statement (IDS) Filed

10684053 - GAU: 2183

PTO/SB/08a (01-10)

Approved for use through 07/31/2012. OMB 0851-0031

U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

INFORMATION DISCLOSURE STATEMENT BY APPLICANT
(Not for submission under 37 CFR 1.99)

Application Number	10684053
Filing Date	2003-10-09
First Named Inventor	Chandan Mathur
Art Unit	2183
Examiner Name	David J. Huisman
Attorney Docket Number	1934-012-03

U.S.PATENTS Remove

Examiner Initial*	Cite No	Patent Number	Kind Code[1]	Issue Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear
	1					

If you wish to add additional U.S. Patent citation information please click the Add button. Add

U.S.PATENT APPLICATION PUBLICATIONS Remove

Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear
	1					

If you wish to add additional U.S. Published Application citation information please click the Add button. Add

FOREIGN PATENT DOCUMENTS Remove

Examiner Initial*	Cite No	Foreign Document Number[3]	Country Code[2]	Kind Code[4]	Publication Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear	T[5]
	1	09-097204	JP		1997-04-08	SUN MICROSYST INC		☐

If you wish to add additional Foreign Patent Document citation information please click the Add button Add

NON-PATENT LITERATURE DOCUMENTS Remove

Examiner Initials*	Cite No	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc), date, pages(s), volume-issue number(s), publisher, city and/or country where published.	T[5]

EFS Web 2.1.17

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /DJH/

Receipt date: 06/01/2010	Application Number	10684053	10684053 - GAU: 2183
INFORMATION DISCLOSURE STATEMENT BY APPLICANT (Not for submission under 37 CFR 1.99)	Filing Date	2003-10-09	
	First Named Inventor	Chandan Mathur	
	Art Unit	2183	
	Examiner Name	David J. Huisman	
	Attorney Docket Number	1934-012-03	

	1		☐

If you wish to add additional non-patent literature document citation information please click the Add button [Add]

EXAMINER SIGNATURE

Examiner Signature	/David Huisman/	Date Considered	11/17/2010

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through a citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

[1] See Kind Codes of USPTO Patent Documents at www.USPTO.GOV or MPEP 901.04. [2] Enter office that issued the document, by the two-letter code (WIPO Standard ST.3). [3] For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. [4] Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST.16 if possible. [5] Applicant is to place a check mark here if English language translation is attached.

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /DJH/

Receipt date: 10/08/2010

Doc code: IDS

Doc description: Information Disclosure Statement (IDS) Filed

10684053 - GAU: 2183

PTO/SB/08a (01-10)

Approved for use through 07/31/2012. OMB 0651-0031

U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

INFORMATION DISCLOSURE STATEMENT BY APPLICANT
(Not for submission under 37 CFR 1.99)

Application Number	10684053
Filing Date	2003-10-09
First Named Inventor	Chandan Mathur
Art Unit	2183
Examiner Name	David J. Huisman
Attorney Docket Number	1934-012-03

U.S.PATENTS Remove

Examiner Initial*	Cite No	Patent Number	Kind Code[1]	Issue Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear
	1	7143418		2006-11-28	PATTERSON	
	2	6282578		2001-08-28	Aizono et al.	

If you wish to add additional U.S. Patent citation information please click the Add button. Add

U.S.PATENT APPLICATION PUBLICATIONS Remove

Examiner Initial*	Cite No	Publication Number	Kind Code[1]	Publication Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear
	1					

If you wish to add additional U.S. Published Application citation information please click the Add button. Add

FOREIGN PATENT DOCUMENTS Remove

Examiner Initial*	Cite No	Foreign Document Number[3]	Country Code[2] i	Kind Code[4]	Publication Date	Name of Patentee or Applicant of cited Document	Pages,Columns,Lines where Relevant Passages or Relevant Figures Appear	T[5]
	1							☐

If you wish to add additional Foreign Patent Document citation information please click the Add button Add

NON-PATENT LITERATURE DOCUMENTS Remove

EFS Web 2.1.17

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /DJH/

Receipt date: 10/08/2010	Application Number	10684053 10684053 - GAU: 2183
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing Date	2003-10-09
(Not for submission under 37 CFR 1.99)	First Named Inventor	Chandan Mathur
	Art Unit	2183
	Examiner Name	David J. Huisman
	Attorney Docket Number	1934-012-03

Examiner Initials*	Cite No	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc), date, pages(s), volume-issue number(s), publisher, city and/or country where published.	T5
	1	TODD GREANIER, Serialization API, July 2000, [online], [retrieved on 2009-06-18]. Retrieved from the Internet , <java.sun.com/developer/technicalArticles/Programming/serialization/>, pages 1-7 as printed.	☐
	2	Ann Wollrath, Roger Riggs, and Jim Waldo, "A Distributed Object Model for the Java System", June 1996, [online], [retrieved on 2009-06-18]. Retrieved from the Internet <usenix.org/publications/library/proceedings/coots96/wollrath.html>, pages 1-14 as printed.	☐
	3	CANADIAN INTELLECTUAL PROPERTY OFFICE, OFFICE ACTION DATED NOVEMBER 23, 2009, FOR CANADIAN APPLICATION NO. 2,503,613 FILED MAY 21, 2004, pages 4.	☐
	4	Korean Intellectual Property Office, Office Action for Korean Patent Application No. 10-2005-7007749 dated March 9, 2010, pages 2.	☐
	5	European Patent Office, Office Action for European Patent Application No. 03 781 551.1-1243 dated December 3, 2010, pages 2.	☐
	6	LECURIEUX-LAFAYETTE G: "Un Seul FPGA Dope Le Traitement D'Images", Electronique, CEP Communication, Paris, FR, no. 55, January 1996, pp. 101-103	☐
	7	AUSTRALIAN PATENT OFFICE - EXAMINER'S REPORT NO. 3 DATED 13 JULY 2010, FOR AUSTRALIAN PATENT APPLICATION NO. 2003287320, pages 3	☐
	8	United States Patent Office, Office Action for United States Patent Application No. 12/151116, dated March 31, 2010, pages 29.	☐
	9	Erich Gamma Richard Helm Ralph Johnson John Vlissides, "Design Patterns Elements of Reusab e Object-Oriented Software", Copyright 1995 by Addison Wesley Longman, Inc, pages 21.	☐
	10	United States Patent Office, Office Action for United States Patent Application No. 10/683929, dated September 24, 2010, pages 40.	☐

EFS Web 2.1.17

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /DJH/

Receipt date: 10/08/2010	Application Number	10684053	10684053 - GAU: 2183
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing Date	2003-10-09	
(Not for submission under 37 CFR 1.99)	First Named Inventor	Chandan Mathur	
	Art Unit	2183	
	Examiner Name	David J. Huisman	
	Attorney Docket Number	1934-012-03	

If you wish to add additional non-patent literature document citation information please click the Add button	Add		
EXAMINER SIGNATURE			
Examiner Signature	/David Huisman/	Date Considered	11/17/2010

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through a citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

[1] See Kind Codes of USPTO Patent Documents at www.USPTO.GOV or MPEP 901.04. [2] Enter office that issued the document, by the two-letter code (WIPO Standard ST.3). [3] For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. [4] Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST.16 if possible. [5] Applicant is to place a check mark here if English language translation is attached.

ALL REFERENCES CONSIDERED EXCEPT WHERE LINED THROUGH. /DJH/

Notice of References Cited	Application/Control No. 10/684,053	Applicant(s)/Patent Under Reexamination MATHUR ET AL.	
	Examiner DAVID J. HUISMAN	Art Unit 2183	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
*	A	US-6,405,266	06-2002	Bass et al.	719/328
	B	US-			
	C	US-			
	D	US-			
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	Oki et al., "The Information Bus - An Architecture for Extensible Distributed Systems", 1993, pp.58-68
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

The Information Bus®– An Architecture for Extensible Distributed Systems

Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen

Teknekron Software Systems, Inc.
530 Lytton Avenue, Suite 301
Palo Alto, California 94301
{boki, pfluegl, alexs, skeen}@tss.com

Abstract

Research can rarely be performed on large-scale, distributed systems at the level of thousands of workstations. In this paper, we describe the motivating constraints, design principles, and architecture for an extensible, distributed system operating in such an environment. The constraints include continuous operation, dynamic system evolution, and integration with extant systems. The *Information Bus*, our solution, is a novel synthesis of four design principles: core communication protocols have minimal semantics, objects are self-describing, types can be dynamically defined, and communication is anonymous. The current implementation provides both flexibility and high performance, and has been proven in several commercial environments, including integrated circuit fabrication plants and brokerage/trading floors.

1 Introduction

In the 1990s, distributed computing has truly moved out of the laboratory and into the marketplace. This transition has illuminated new problems, and in this paper we present our experience in bringing large-scale, distributed computing to mission-critical applications. We draw from two commercial application areas: integrated circuit (IC) fabrication plants and brokerage/trading floors. The system we describe in this paper has been installed in over one hundred fifty production sites and on more than ten thousand workstations. We have had a unique opportunity to observe distributed computing within the constraints of commercial installations and to draw important lessons.

This paper concentrates on the problems posed by a "24 by 7" commercial environment, in which a distributed system must remain operational twenty-four hours a day, seven days a week. Such a system must tolerate software and hardware crashes; it must continue running even during scheduled maintenance periods or hardware upgrades; and it must be able to evolve and scale gracefully without affecting existing services. This environment is crucially important to our customers as they move toward real-time decision support and event-driven processing in their commercial applications.

One class of customers manufactures integrated circuit chips. An IC factory represents such an enormous investment in capital that it must run twenty-four hours a day. Any down time may result in a huge financial penalty from both lost revenue and wasted materials. Despite the "24 by 7" processing requirement, improvements to software and hardware need to be made frequently.

Another class of customers is investment banks, brokers, and funds managers that operate large securities trading floors. Such trading floors are very data-intensive environments and require that data be disseminated in a timely fashion to those who need it. A one-minute delay can mean thousands of dollars in lost profits. Since securities trading is a highly competitive business, it is advantageous to use the latest software and hardware. Upgrades are frequent and extensive. The system, therefore, must be designed to allow seamless integration of new services without affecting existing services.

In the systems that we have built and installed, dynamic system evolution has been the greatest challenge. The sheer size of these systems, which can consist of thousands of workstations, requires novel solutions to problems of system evolution and maintenance. Solving these problems in a large-scale, "24 by 7" environment leads to more than just quantitative differences in how systems are built—these solutions lead to fundamentally new ways of organizing systems.

The contributions of this paper are two-fold. One is the description of a set of system design principles that were crucial in satisfying the stringent requirements of "24 by 7" environments. The other is the demonstration of the usefulness and validity of these principles by discussing a body of

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
SIGOPS '93/12/93/N.C., USA
© 1993 ACM 0-89791-632-8/93/0012...$1.50

software out in the field. This body of software consists of several tools and modules that use a novel communications infrastructure known as the Information Bus. All of the software components work together to provide a complete distributed system environment.

This paper is organized as follows. Section 2 provides a detailed description of the problem domain and summarizes the requirements for a solution. Section 3 outlines the Information Bus architecture in detail, states principles that drove our design, and discusses some aspects of the implementation. Section 4 describes the notion of adapters, which mediate between old applications and in the Information Bus. Section 5 describes other software components that use the Information Bus and provide a complete application environment. This section also provides an example to illustrate the system. Section 6 presents related work. Section 7 summarizes the paper and discusses open issues. The Appendix discusses the performance characteristics of the Information Bus.

2 Background

An IC fabrication plant represents a huge capital investment. This investment, therefore, is cost-effective only if it can remain operational twenty-four hours a day. To bring down an entire plant in order to upgrade a key software component, such as the "Work-In-Process" tracking system, would result in lost revenue and wasted material. There is no opportunity to "reboot" the entire system. We state this requirement as R1:

R1 *Continuous operation.* It is unacceptable to bring down the system for upgrades or maintenance.

Despite the need for continuous operation, frequent changes in hardware and software must also be supported. New applications and new versions of existing applications need to be brought on-line. Business requirements and factory models change, and such changes need to be reflected in the application behavior. For example, new equipment types could be introduced into the factory. We state this requirement as R2:

R2 *Dynamic system evolution.* The system must be capable of adapting to changes in application architecture and in the type of information exchanged. It should also support the dynamic integration of new services and information.

In the systems that we have built and installed, this requirement has posed the greatest challenge. The sheer size of these systems, typically ranging from one hundred to a thousand workstations, makes changes expensive or even impossible, unless change is planned from the beginning.

Businesses often have huge outlays in existing hardware, software, and data. To be accepted by the business community, a new system must be capable of leveraging existing technology; an organization will not throw away the product of an earlier costly investment. We state this requirement as R3:

R3 *Legacy systems.* New software must be able to interact smoothly with existing software, regardless of the age of that software.

Other important requirements are fault-tolerance, scalability, and performance. The system must be fault-tolerant; in particular it must not have a single point of failure. The system must scale in terms of both hardware and data. Finally, our installations must meet stringent performance standards. In this paper, we focus on requirements R1, R2, and R3 because they represent "real-world" constraints that have been less studied in research settings.

The typical customer environment consists of a distributed collection of independent processors, *nodes*, that communicate with each other by passing messages over the network. Nodes and the network may fail, and it is assumed that these failures are fail-stop [Schneider83] and not Byzantine [Lamport82].[1] The network may lose, delay, and duplicate messages, or deliver messages out of order. Link failures may cause the network to partition into subnetworks that are unable to communicate with each other. We assume that nodes eventually recover from crashes. For any pair of nodes, there will eventually be a time when they can communicate directly with each other after each crash.

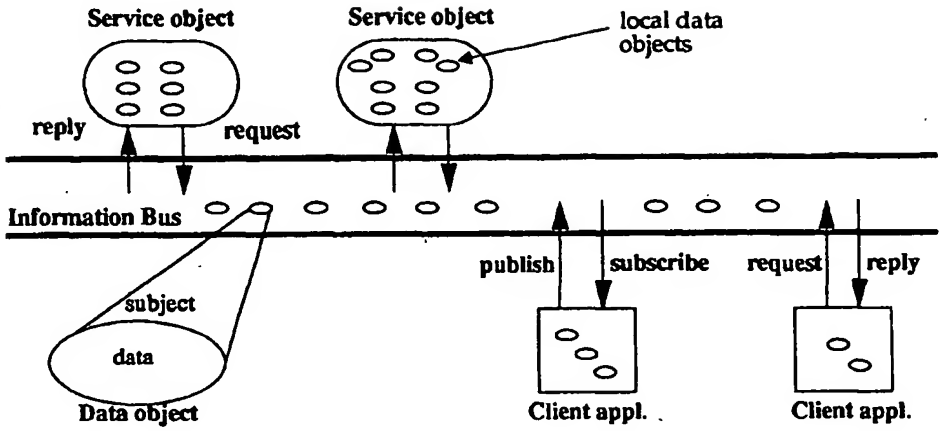
3 Information Bus Architecture

The requirements of a "24 by 7" environment dictated numerous design decisions that ultimately resulted in the Information Bus that we have today. We have distilled those decisions into several design principles, which are highlighted as they become apparent in this section.

Because it is impossible to anticipate or dictate the semantics of future applications, it is inadvisable to hard-code application semantics into the core communications software—for performance reasons and because there is no "right" answer for every application [Cheriton93]. For example, complex ordering semantics on message delivery are not supported directly. Atomic transactions are also not supported: in our experience most applications do not need the strong consistency requirements of transactions. Instead, we provide tools and higher-level services to cover the range of requirements. This allows us to keep the communications software efficient, while still allowing us to adapt to the specific needs of each class of customers. The following principle is motivated by requirements R2 and R3.

1. Failures in our customer environments closely approximate fail-stop behavior; furthermore, protecting against the rare Byzantine failure is generally too costly.

FIGURE 1. Model of computation



P1 *Minimal core semantics.* The core communication system and tools can make few assumptions about the semantics of application programs.

Our model of computation consists of objects, both data objects and service objects, and two styles of distributed communication: traditional request/reply (using remote procedure call [Birrell84]) and publish/subscribe, the hallmark of the Information Bus architecture. This is depicted in Figure 1. Publish/subscribe supports an event-driven communication style and permits data to be disseminated on the bus, whereas request/reply supports a demand-driven communication style reminiscent of client/server architectures. For each communication style, there are different levels of *quality of service*, which reflect different design trade-offs between performance and fault-tolerance. The next section elaborates on these mechanisms.

An *object* is an instance of a class, and each class is an implementation of a type[2]. Our system model distinguishes between two different kinds of objects: *service objects* that control access to system resources and *data objects* that contain information and that can easily be transmitted. A *service object* encapsulates and controls access to resources such as databases or devices and its local data objects. Service objects typically contain extensive state and may be fault-tolerant. Because they tend to be large-grained, they are not easily marshalled into a wire format and transmitted. Instead of migrating to another node, they are invoked where they reside, using a form of remote procedure call. Examples of service objects include network file systems, database systems, print services, and name services.

A *data object*, on the other hand, can be easily copied, marshalled, and transmitted. Such objects are at the granularity of typical C++ objects or database records. They abstract and encapsulate application-level concepts such as documents, bank accounts, CAD designs, and employee records. They run the gamut from abstracting simple data records to defining complex behaviors, such as "recipes" for controlling IC processing equipment.

Each data object is labelled with a *subject* string. Subjects are hierarchically structured, as illustrated by the following well-formed subject "fab5.cc.litho8.thick." This subject might translate to plant "fab5," cell controller, lithography station "litho8," and wafer thickness. Subjects are chosen by applications or users.

The second principle, P2, is motivated by requirements R2 and R3, and together with data abstraction, allows applications to adapt automatically to changes in an object's implementation and data representation.

P2 *Self-describing objects.* Objects, both service and data objects, are "self-describing." Each supports a *meta-object protocol* [Kiczales91], allowing queries about its type, attribute names, attribute types, and operation signatures.

P2 enables our systems and applications to support introspective access to their services, operations, and attributes. In traditional environments, introspection is used to develop program analysis tools, such as class browsers and debuggers. In the Information Bus environment, introspection is used by applications to adapt their behaviors to change. This is key to building systems that can adapt to change at run-time.

2. A type is an abstraction whose behavior is defined by an *interface* that is completely specified by a set of *operations*. Types are organized into a supertype/subtype hierarchy. A class is an implementation of a type. Specifically, a class defines methods that implement the operations defined in a type's interface.

Introspection enables programmers to write generic software that can operate on a wide range of types. For example, consider a "print" utility. Our implementation of this utility can accept any object of any type and produce a text description of the object. It examines the object to determine its type, and then generates appropriate output. In the case of a complex object, the utility will recursively descend into the components of the object. The print utility only needs to understand the fundamental types, such as integer or string, but it can print an object of any type composed of those types.

The third principle, P3, enables new concepts and abstractions to be introduced into the system.

P3 *Dynamic classing*. New classes implementing either existing or new types can be dynamically defined and used to create instances. This is supported for both service and data objects.

P3 enables new types to be defined, on-the-fly. Note that P2 enables existing applications to make use of these new types without re-programming or re-linking.

To support dynamic classing, we have implemented TDL, a small, interpreted language based on CLOS [Keene89]. We have chosen a subset of CLOS that supports a full object model, but that could be supported in a small, efficient run-time environment.

3.1 Publish/Subscribe Communication

To disseminate data objects, data producers generate them, label them with an appropriate subject, and *publish* them on the Information Bus. To receive such objects, data consumers *subscribe* to the same subject. Consumers need not know who produces the objects, and producers need not know who consumes or processes the objects. This property is expressed in principle P4. We call this model *Subject-Based Addressing*™, and it is a variant of a generative communication model [Carriero89]. This principle is motivated by requirements R1 and R2, and it allows applications to tolerate architectural changes on the fly.

P4 *Anonymous communication*. Data objects are sent and received based on a subject, independent of the identities and location of data producers and data consumers.

Subjects can be partially specified or "wildcarded" by the consumer, which permits access to a large collection of data from multiple producers with a single request. The Information Bus itself enforces no policy on the interpretation of subjects. Instead, the system designers and developers have the freedom and responsibility to establish conventions on the use of subjects.

Anonymous communication is a powerful mechanism for adapting to software changes that occur at run-time. A new subscriber can be introduced at any time and will start receiving immediately new objects that are being published under the subjects to which it has subscribed. Similarly, a new publisher can be introduced into the system, and existing subscribers will receive objects from it. Our model of computation does not require a traditional name service like Sun's NIS or Xerox's Clearinghouse [Oppen83].

In a traditional distributed system, whenever new services are added to the system, or a service is being replaced with a new implementation, the name service must be updated with the new information. To use that information, all applications must be aware that the new services exist, must contact the name service to obtain the location of the new service, and then bind to the service. In our model, the new implementation need only use the same subjects as the old implementation; neither publishers nor subscribers must be aware of the change. Subject names can be rebound at *any* time to a new address, a facility that is more general than traditional late-binding.

The semantics of publish/subscribe communication depends on the requirements of the application. The usual semantics we provide is *reliable* message delivery. Under normal operation, if a sender and receiver do not crash and the network does not suffer a long-term partition, then messages are delivered exactly once in the order sent by the same sender; messages from different senders are not ordered. If the sender or receiver crashes, or there is a network partition, then messages will be delivered at most once.

A stronger semantics is *guaranteed* message delivery. In this case, the message is logged to non-volatile storage before it is sent. The message is guaranteed to be delivered at least once, regardless of failures. The publisher will retransmit the message at appropriate times until a reply is received. If there is no failure, then the message will be delivered exactly once. Guaranteed delivery is particularly useful when sending data to a database over an unreliable network.

For local area networks, reliable publication is implemented with Ethernet broadcast. This choice allows the same data to be delivered to a large number of destinations without a performance penalty. Moreover, Ethernet broadcast eliminates the need for a central communication server. Our current implementation uses UDP packets in combination with a retransmission protocol to implement reliable delivery semantics.

In our implementation of subject-based addressing, we use a daemon on every host. Each application registers with its local daemon, and tells the daemon to which subjects it has subscribed. The daemon forwards each message to each application that has subscribed. It uses the subject contained in the message to decide which application receives which message.

Given the high traffic rates, Ethernet broadcast across wide area networks is undesirable. We could use IP multicast [Cheriton85], but unfortunately, commercial implementations are not mature enough for mission-critical use. Therefore, wide area networks require additional communication tools.

Our implementation uses application-level "information routers" to solve the problem posed by wide area networks. To the Information Bus, these routers look like ordinary applications, but they actually integrate multiple instances of the bus. Messages are received by one router using a subscription, transmitted to another router, and then re-published on another bus. The router is intelligent about which messages are sent to which routers: messages are only re-published on buses for which there exists a subscription on that subject; the router can also perform other functions, such as transforming subjects or logging messages to non-volatile storage. Thus, the overall effect is to create the illusion of a single, large bus that is capable of publishing over any network.

3.2 Dynamic Discovery

In a distributed system, it is often necessary for an application to discover the identify of the participants in a protocol. For example, a new client needs to determine the set of servers that serve a subject; a new server needs to determine if any clients have pending requests; a replicated server needs to find the other servers that maintain the replicated data. Specifically, in Xerox's corporate email service, a traditional distributed system, client mail applications find a mail service for posting mail messages by using an expanding ring broadcast technique, a kind of discovery protocol [Xerox88].

In the Information Bus, the discovery protocol is in the form of two publications. One participant publishes "Who's out there?" under a subject. The other participants publish "I am" and other information describing their state, if they serve the subject in question. Section 3.3 provides a specific example of this exchange. This approach preserves P4 (anonymous communication). The subject alone is enough for one participant to make contact with its cohorts.

The publish/subscribe communication model is well-suited to supporting a discovery protocol. Since publication does not require any boot-strapping or name resolution, it can be the first step in a protocol. We are effectively using the network itself as a name service. A subject is mapped to a specific set of servers by allowing the servers to choose themselves. The "Who's out there?" publication can contain service-specific information, so further refinements are possible when selecting servers.

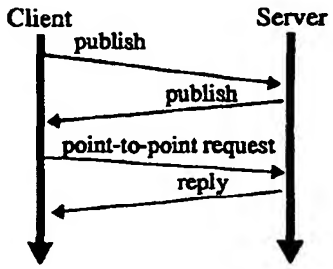
3.3 Remote Method Invocation

Remote method invocation (RMI), or remote procedure call, is the second means of performing distributed computations. This paired request/reply satisfies the demand-driven style of interaction. Clients invoke a method on a remote server object without regard to that server object's location, the server object executes the method, and the server replies to the client. Servers are named with subjects.

Standard RMI provide *exactly-once* semantics under normal operation and *at-most-once* semantics in the presence of failures. Customer-specific requirements such as *exactly-once* semantics, which guarantees that the method will be executed exactly once, even in the presence of failures, can be built on the a layer above standard RMI.

There are two parts to RMI: discovering the server object for a client, and establishing a connection to that server over which requests and replies will flow. The discovery algorithm in our implementation employs publish/subscribe communication as described in Section 3.2. In this algorithm, the client searches for all servers by publishing a query message on a subject specific to that service. The servers receive this message, and then they publish their point-to-point address to all clients on the same subject. Finally, the client invokes a service request on a server object using the point-to-point address. The point-to-point address can refer to any simple, connection mechanism, such as a TCP/IP connection [Postel81]. Figure 2 illustrates this protocol.

FIGURE 2. RMI Protocol



More than one server can respond to requests on a subject. Several server objects can be used to provide load balancing or fault-tolerance. Our system allows an application to choose between several different policies. The servers can decide among themselves which one will respond to a request from the client. Alternatively, the client can receive every response from all of the servers and then decide which server the client wants to use.

4 Adapters

The Information Bus must allow for interaction with existing systems, as dictated by requirement R3. To integrate existing applications into the Information Bus we use

software modules called *adapters*. These adapters convert information from the data objects of the Information Bus into data understood by the applications, and vice versa. Adapters must live in two worlds at once, translating communication mechanisms and data schemas. Adapters often require P1 in order to be feasible.

Adapters are essential for integrating the Information Bus into a commercial environment. In the factory floor example, our customer already had a Work In Progress (WIP) system with its own data schemas. We designed an adapter that allows the existing WIP software to communicate with the Information Bus. This achievement demonstrates the flexibility of the Information Bus model: the existing WIP system is written in Cobol, and there is only a primitive terminal interface. The adapter must act as a virtual user to the terminal interface.

The Object Repository is an example of a sophisticated adapter that integrates a commercially available relational database system into the Information Bus architecture. The Object Repository maps Information Bus objects into data base relations for storage or retrieval. This mapping is driven by the meta-data of each object. Besides satisfying requirement R3 as an adapter, the design of the repository also supports dynamic system evolution, which satisfies requirement R2. Users may work freely in the object model without concerning themselves with the relational data model[3] [Codd70]. Using P2, the repository can automatically adapt the relational model to the type structure of the data objects.

The repository behaves as a kind of schema converter from objects to database tables, and vice versa. Users are thus insulated from any changes implementors may wish to make to the database representation of objects. For example, our conversion algorithm decomposes a complex object into one or more database tables and reconstructs a complex object from one or more database tables to answer a query from a user. This conversion respects the type hierarchy, enabling queries to return all objects that satisfy a constraint, including objects that are instances of a subtype. Old queries will still work even as new subtypes are introduced, which helps to satisfy R2. This operation can be fully automated; only the type information is necessary to do the transformation. When the repository needs to store an instance of a previously unknown type, it is capable of generating one or more new database tables to represent the new type.

3. Our object model differs significantly from the relational data model in the following way. A database table is a flat structure composed of simple data types and has little semantics, while an object may contain other objects, may have subtypes or supertypes, and may have methods to manipulate instances of the type.

The repository may be configured in any number of ways, depending on the application. For example, it may be configured as a capture server that captures all objects for a given set of subjects and inserts those objects automatically into the repository under those subjects; it may also be configured as a query server to receive requests from clients and return replies.

5 Example Application

In the previous section, we discussed the Information Bus architecture primarily in terms of an abstract object model and two communication styles, and we espoused several principles of system design. To make the architecture more concrete, we present an example that shows how the various components fit together into a single application and that illustrates how an application can adapt to changes in the environment. In particular, we show how the principles are applied in the context of the example, and discuss some software components that have been built on top of the Information Bus and that are installed at customer sites.

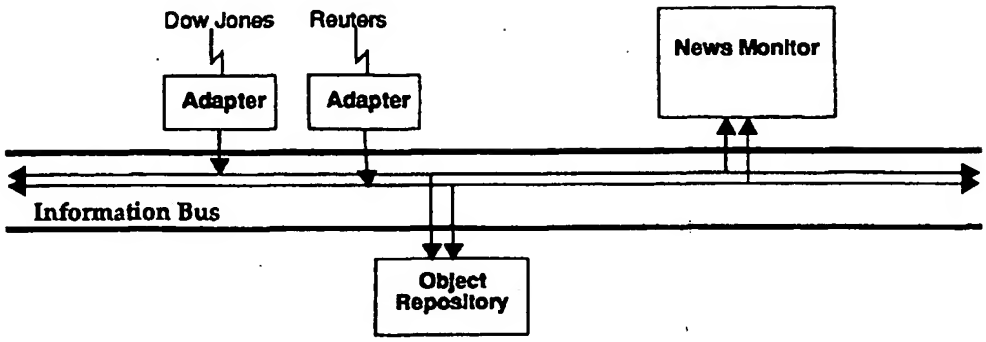
Figure 3 shows an example taken from a financial trading floor, where traders must receive news stories in near real-time. Two news adapters receive news stories from communication feeds connected to outside news services, such as Dow Jones and Reuters. Each raw news service defines its own news format. Each adapter parses the received data into an appropriate vendor-specific subtype of a common Story supertype, and publishes each story on the Information Bus under a subject describing the story's primary topic (for example, "news.equity.gmc" for stories on General Motors). P1 ensures that the raw feeds do not have to support complex semantics.

The figure depicts two applications that consume the Story objects: the Object Repository and the News Monitor. The News Monitor subscribes to and displays all stories of interest to its user. Incoming stories are first displayed in a "headline summary list." This list format is defined by a "view" that specifies a set of named attributes[4] from incoming objects and formatting information. When the user selects a story in the summary list, the entire story is displayed. This is accomplished by using the object's meta-data to iterate through all of its attributes and display them, as provided by P2.

The Object Repository subscribes to all news stories and inserts them into a relational database. The repository converts Story objects into a database table format. This conversion is nontrivial because a story is a highly structured object containing other objects such as lists of "industry groups," "sources," and "country codes." Every object

4. "Attributes" of an object are often referred to as "instance variables" or "fields."

FIGURE 3. Brokerage Trading Floor



must be mapped into collections of simple database relations.

5.1 Graphical Application Builder

We needed a simple, general way to access information on the Information Bus and in the database in a pleasing, graphical form. It was not satisfactory to build a single, static solution, since each customer has different needs, and they change frequently. Instead, we built a graphical application builder, designed for applications with a graphical user interface builder. We have used it for several applications, including the News Monitor example and the front-end to a Factory Configuration System, which is the system for storing factory control information.

The application builder is an interpreter-driven, user interface toolkit. It combines the ability to construct sophisticated user interfaces with a simple, object-oriented language. All high-level application behavior is encoded in the interpreted language; only low-level behavior that is common to many applications is actually compiled.

The resulting applications are fully integrated into the Information Bus, providing access to all subjects and services. It is possible to examine the list of available services on the Information Bus by using various name services. Services are self-describing, so users can inspect the interface description for each service. Using that information, a user can quickly construct a basic user interface for any service. This whole process requires only a few minutes, and typically no compilation is involved. Sometimes, a single user interface can be used to access several services, further reducing the amount of work involved.

5.2 Dynamic System Evolution

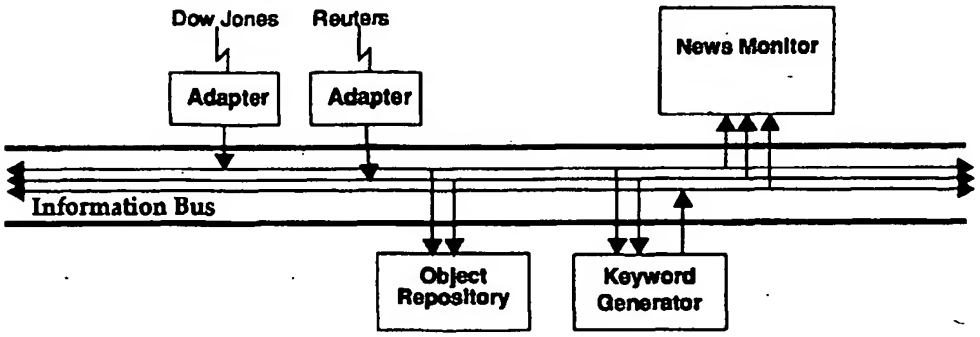
In this section we illustrate how our design principles support the requirement of dynamic system evolution. First, we consider the introduction of a new type into the Information Bus (P3). The Object Repository can dynamically generate extensions to the database schema to accommodate such new types. Such generation may entail creating one or more new database tables for each new type, depending on the particular database representation. When instances of the new type are received, they are dynamically converted to the new database schema. P2 implies that the repository will be able to recognize and process objects that have a new type.

Second, we consider what happens if a new service of a completely different nature is introduced. Consider a Keyword Generator, as illustrated in Figure 4. The Keyword Generator subscribes to stories on major subjects and searches the text of each story for "keywords" that have been designated under several major "categories." For each Story object, a list of keywords is constructed as a named Property[5] object of the Story object and published under the same subject. It also supports an interactive interface that allows clients to browse categories and associated keywords.

When the Keyword Generator comes on-line, the News Monitor will start receiving Property objects on the same subjects on which it is receiving Story objects. According to P4, the News Monitor will be able to receive the new data immediately. Since properties are a general concept in the architecture, it can reasonably be assumed the News Monitor is configured to accept Property objects, to associate them with the objects they reference, and to display them along with the attributes of an object when the object is selected. This capability can be set up using the scripting language of the Graphical Application Builder,

5. The Object Management Group's "Object Services Architecture" is the basis for the nomenclature used here. Accordingly, a "property" is a name-value pair that can be dynamically defined and associated with an object. In this example, the property name is "keywords" and the value is the set of keywords found.

FIGURE 4. Adding a Key Word Generator to the Brokerage Trading Floor.



The interactive interface of the Keyword Generator is an instance of a new service type. Using introspection, the News Monitor can enable the user to interact with this new type: menus listing the operations in the interface can be popped up, and dialogue boxes that are based on the operations' signatures can lead the user through interactions with the new service.

Hence, as soon as the Keyword Generator service comes on-line, the user's world becomes much richer, both in terms of information and of services. As seen by the user, the new service and information is dynamically integrated into the environment. This shows the adaptive flexibility of the overall approach. Note that the example requires only that the News Monitor support Property objects, but it does not require knowledge of how properties are generated, in compliance with P4.

6 Related Work

The Linda system, developed at Yale University, was the first system to support a generative communication model [Carriero89]. In Linda, processes generate *tuples*, which are lists of typed data fields. These generated tuples are stored in *tuple space* and persist until explicitly deleted. Other Linda processes may invoke operations to remove or read a tuple from tuple space. Storing a tuple in tuple space, in effect, is like one process "broadcasting" a tuple to many other processes.

A key difference between Linda and the Information Bus is the data model. Linda tuples are data records, not objects. Moreover, Linda does not support a full meta-object protocol. Self-describing objects have been invaluable in enabling data independence, the creation of generic data manipulation and visualization tools, and achieving the system objective of permitting dynamic integration of new services.

Another key difference between Linda and the Information Bus is the mechanism for accessing data. Linda accesses data based on attribute qualification, just as relational databases do. Though this access mechanism is more powerful than subject-based addressing, we believe that it is more general than most applications require. We have found that subject names are quite adequate for our needs, and they are far easier to implement than attribute qualification. We also argue that subject-based addressing scales more easily, and has better performance, than attribute qualification.

In the ISIS system [Birman89] processes may join *process groups*, and messages can be addressed to every member of a process group. ISIS has focused on various message delivery semantics without regard for application-level semantics. Hence, it does not support a high-level object model.

Usenet [Fair84] is the best known example of a large-scale communications system. A user may post an *article* to a *news group*. Any user who has subscribed to that news group will eventually see the article. Usenet, however, makes no guarantees about message delivery: messages can be lost, duplicated, or delivered out of order. Delivery latency can be very large, on the order of weeks in some cases. On the other hand, Usenet moves an impressive volume of data to a huge number of sites.

Usenet should be viewed as a communication system, whose focus is moving text among humans. News articles are unstructured, and no higher-level object model is supported. It would make an unsuitable communications environment for our customer's applications, given its weak delivery semantics and long latencies.

The Zephyr notification service [DellaFera88], developed at MIT as part of Project Athena [Balkovich85], is used by applications to transport time-sensitive textual information asynchronously to interested clients in a distributed workstation environment. The notice subscription service layer is of particular interest because it most resembles our publish/subscribe communication model. In Zephyr, a

client interested in receiving certain classes of messages, registers its interest with the service. The service uses "subscription multicasting" (their term) to compute dynamically the set of subscribers for a particular class of message and sends a copy of the message only to those recipients that have subscribed.

This subscription multicasting mechanism relies heavily on a centralized location database that maps unique Zephyr IDs to information like geographical location and host IP address, and it is not at all clear how well such an implementation would work in a wide-area network. Furthermore, this mechanism is inefficient if the number of interested clients is very large.

7 Conclusion

In this paper, we described the requirements posed by a "24 by 7" commercial environment, such as the factory floor automation system of a semiconductor fabrication plant. The centerpiece of our solution is the Information Bus. The Information Bus has been ported to most desktop and server platforms, and has been installed at more than one hundred fifty sites around the world, running on over ten thousand hosts. We have demonstrated that this architecture is a successful approach to building distributed systems in a commercial setting.

Minimal core semantics (P1), self-describing objects (P2), a dynamic classing system (P3), and anonymous communication (P4) allow applications that use the Information Bus to evolve gracefully over time. P1 prevents applications from being crippled by the communication system. P2 allows new types to be handled at run-time. P3 enables new types to be introduced without recompilation. P4 permits new modules to be transparently introduced into the environment.

The first requirement was continuous availability (R1). Anonymous communication (P4) allows a new service to be introduced into the Information Bus. A new server that implements such a service can transparently take over the function of an obsolete server. The old server can be taken off-line after it has satisfied all of outstanding requests. With this technique, software upgrades can be performed on a live system. In addition, new services can be offered at any time, and existing clients can take advantage of these new services.

The second requirement was support for dynamic system evolution (R2). New services and new types can be added to the Information Bus without affecting existing services or types. Self-describing data (P2) ensures that the data model and data types can be substantially enhanced without breaking older software. In many cases, older software can make use of the enhancements in the data objects immediately. This ability implies that applications can provide additional functionality by only changing the data model.

The third requirement was the ability to integrate legacy systems (R3). The Information Bus connects to legacy systems through the use of adapters (Section 4), which mediate between other systems and the Information Bus. The principle of minimal core semantics (P1) aids in the construction of adapters.

Acknowledgments

The authors wish to thank the anonymous referees for their many helpful comments. We thank Mendel Rosenblum for passing on to us the program committee's comments that greatly strengthened the final version of the paper. Our thanks also go to Tommy Joseph, Richard Koo, Kieran Harty, Andrea Wagner, and Brendon Whateley. Finally, we thank TSS management, and Dr. JoMei Chang in particular, for their patience during the lengthy preparation of this paper.

References

[Balkovich85] Balkovich, E., S.R. Lerman, and R.P. Parmele. "Computing In Higher Education: The Athena Experience," *Communications of the ACM 28*, 11 (November 1985), pp. 1214–1224.

[Birman89] Birman, Ken, and Thomas Joseph. "Exploiting Replication in Distributed Systems," *in Distributed Systems*, Mullender, Sape, editor, Addison-Wesley, 1989, pp. 319-365.

[Birrell84] Birrell, Andrew D. and Bruce J. Nelson. "Implementing Remote Procedure Calls." *ACM Transactions on Computer Systems 2*, 1(February, 1984), pp. 39–59.

[Carriero89] Carriero, Nicholas and David Gelernter. "Linda in Context". *Communications of the ACM 32*, 4 (April, 1989), pp. 444–458.

[Cheriton85] Cheriton, David R. and Steven E. Deering. "Host Groups: a multicast extension for datagram internetworks." In *Proceedings of the 9th Data Communications Symposium, ACM SIGCOMM Computer Communications Review 15*, 4 (September 1985), pp. 172-179.

[Cheriton93] Cheriton, David R. and Dale Skeen. "Understanding the Limitations of Causally and Totally Ordered Communication." In *Proc. of the 14th Symp. on Operating Systems Principles*, Asheville, North Carolina, December 1993.

[Codd70] Codd, E. F. "A Relational Model for Large Shared Data Banks." *Communications of the ACM 13*, 6 (June, 1970).

[DellaFera88] DellaFera, C. Anthony, Mark W. Eichin, Robert S. French, David C. Jedlinsky, John T. Kohl, and William E. Summersfeld. "The Zephyr Notification Service," *Usenix Conference Proceedings*, Dallas, Texas (February 1988).

[Fair84] Erik Fair, "Usenet, Spanning the Globe." *Unix/World*, 1 (November, 1984), pp. 46-49.

[Lamport82] Lamport, Leslie, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem." *ACM Transactions on Programming Languages and Systems 4*, 3 (July 1982), pp. 382-401.

[Keene89] Keene, Sonya. *Object-Oriented Programming in Common Lisp: A Programmer's Guide to CLOS*, Addison-Wesley, 1989.

[Kiczales91] Kiczales, Gregor, Jim des Rivieres, and Daniel Bobrow. *The Art of the Metaobject Protocol*, MIT Press, 1991.

[Oppen83] Oppen, Derek C. and Y. K. Dalal. "The Clearinghouse: A decentralized agent for locating named objects in a distributed environment." *ACM Tranactions on Office Information Systems 1*, 3 (July 1983), pp. 230-253.

[Postel81] Postel, Jon, "Internet Protocol - DARPA Internet Program Protocol Specification," *RFC 791*, Network Information Center, SRI International, Menlo Park, CA, September 1981.

[Schneider83] Schneider, Fred. "Fail-Stop Processors." *Digest of Papers from Spring CompCon '83 26th IEEE Computer Society International Conference*, March 1983, pp. 66-70.

[Skeen92] Skeen, Dale, "An Information Bus Architecture for Large-Scale, Decision-Support Environments," *Unix Conference Proceedings*, Winter 1992, pp. 183-195.

[Xerox88] Mailing Protocols. Xerox System Integration Standard (May 1988), XNSS 148805.

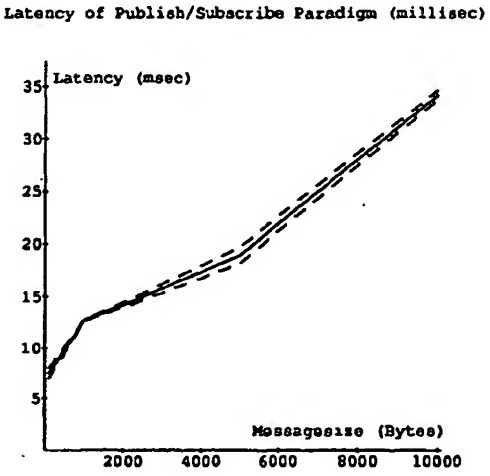
Appendix

For some of our customers, the Information Bus must handle thousands of nodes with thousands of publishers, consumers, clients, and servers. Therefore, performance is crucial. In this section, we measure the performance of the publish/subscribe communication model.

The two factors that most characterize performance are *throughput*, measured in messages per second or bytes per second, and *latency*, measured in seconds. Latency is the average time between the sending of a message and its receipt. In this appendix we present several figures illustrating the performance of the publish/subscribe communication model. The key parameters for performance are the message size and the number of data consumers. Hence, we will plot the throughput and latency versus message size in bytes and explain the effect of the number of consumers.

All data presented here was collected on our development network of Sun SPARCstation 2s and Sun IPXs with twenty-four to forty-eight megabytes of memory running SunOS 4.1.1. The network was a 10 Megabits/second Ethernet, and it was lightly loaded. Since all monitored publishers/consumers are on the same subnet, information does not need to go through any bridges or routers. All message delivery is reliable but not guaranteed. For any given test run, the message size was constant. For the performance data shown here, publishers and consumers were spread over fifteen nodes.

FIGURE 5. Latency vs. Msg Size



The data for Figure 5 was collected by executing one publisher publishing under a single subject. The information is consumed by fourteen consumers (one consumer per node). It shows that the latency depends on the message size. Although not shown, the latency is independent of the number of consumers. The 99%-confidence interval is pre-

sented with dashed lines. The Information Bus has a batch parameter that increases throughput by delaying small messages, and gathering them together. When measuring the latency, the batch parameter was turned off to avoid intentionally delaying the publications. Variances of the data sets used in Figure 5 ranged from 1.1×10^{-4} to 1.7×10^{-2} milliseconds.

FIGURE 6. Throughput - Msgs/Sec vs. Msg Size

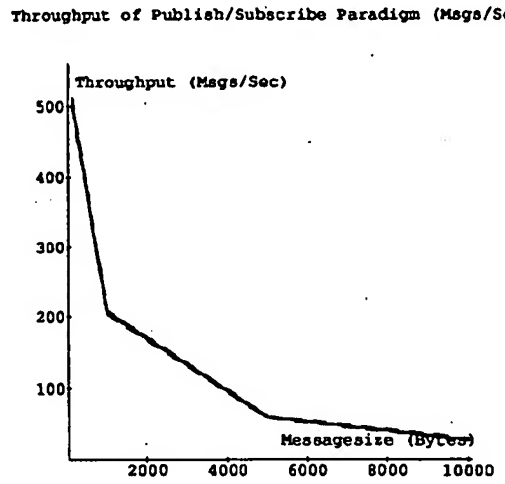


FIGURE 7. Throughput - Bytes/Sec vs. Msg Size

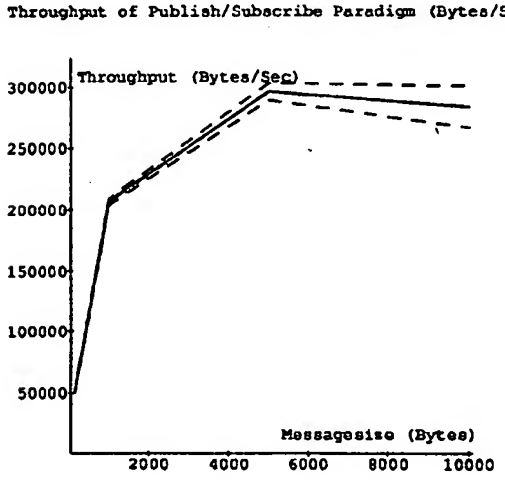
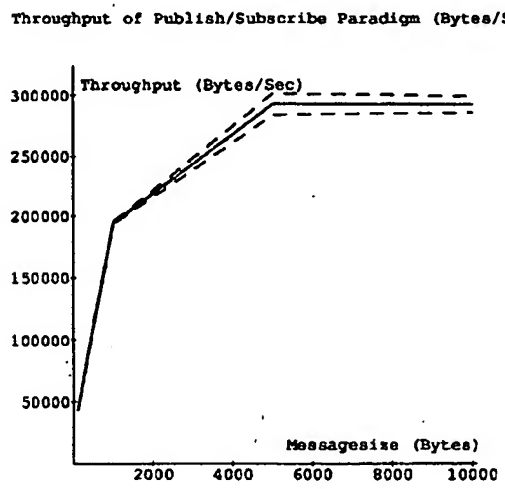


Figure 6 and 7 show the throughput of a network with one publisher publishing under one subject, sending to fourteen consumers. For this test, the batching parameter was turned on. For messages larger than five thousand bytes, the device bandwidth becomes the limiting factor: it is difficult to drive more then 300 Kb/sec through Ethernet with a raw UDP socket, suggesting that the Information Bus represents a low overhead. The slight decrease in throughput and increase in variance between five thousand and ten thousand-byte messages is due to collisions from unrelated network activity.

This set of test cases also verified that the publication rate is independent of the number of subscribers. Therefore, the cumulative throughput over all subscribers is proportional to the number of subscribers. The variances of the data sets used in Figure 6 ranged from 0.25 to 125 messages/second. Figure 7 was plotted based on the same data.

FIGURE 8. Throughput - Effect of the Number of Subjects



The difference between the environment of Figure 7 and Figure 8 is that the publisher published on ten thousand different subjects instead of one, and the fourteen consumers subscribed to all ten thousand subjects. As the data in Figure 8 shows, the number of subjects has an insignificant influence on the throughput. For Figure 8, we collected data in messages/second. These data sets have a variance that ranges from 1.2 to 4.6 messages/second. The time to process each subscription request is not shown in the above figure since these requests are performed once at start-up time.